New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Written Homework 3.
Due Tuesday, May 6th, 2025, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually.

Problem 1: Kernels for Shifted Images (20pts)

In class we discussed why the Gaussian kernel is a better similarity metric for MNIST digits than the inner product. Here we consider an additional modification to the Gaussian kernel.

For illustration purposes we consider 5x5 black and white images: a pixel has value 1 if it is white and value 0 if it is black. For example, consider the following images of two 0s and two 1s:

$$I_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Let $\mathbf{x}_i \in \{0,1\}^{25}$ denote the vectorized version of image I_i, obtained by concatenating the rows of the matrix representation of the image into a vector. Compute the 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using the standard Gaussian kernel $k_G(I_i, I_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}$ (this is just a simple calculation).

(b) Suppose I_1 and I_2 are in our training data and I_3 and I_4 are in our test data. Which training image is most similar to each of our test images according to Gaussian kernel similarity? Do you expect a kernel classifier (k-NN, kernel logistic regression, etc.) to correctly or incorrectly classify I_3 and I_4?

(c) Consider a "left-right shift" kernel, which is a similarity measure defined as follows:

For an image I_i, let I_i^{right} be the image with its far right column removed and let I_i^{left} be the image with its far left column removed. Intuitively, I_i^{right} corresponds to the image shifted one pixel to the right and I_i^{left} corresponds to the image shifted one pixel left. Define a new similarity metric k_{shift} as follows:

$$k_{\text{shift}}(I_i, I_j) = k_G(I_i^{\text{right}}, I_j^{\text{right}}) + k_G(I_i^{\text{left}}, I_j^{\text{left}}) + k_G(I_i^{\text{right}}, I_j^{\text{left}}) + k_G(I_i^{\text{left}}, I_j^{\text{right}})$$

Intuitively this kernel captures similarity between images which are similar *after a shift*, something the standard Gaussian kernel does not account for.

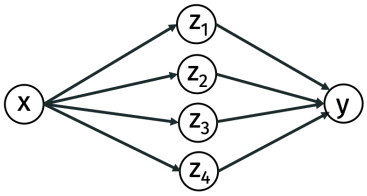
Recompute the a 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using k_{shift}.

(d) Again I_1 and I_2 were in our training data and I_3 and I_4 were in our test data. Now which training image is most similar to each of our test images according to the "left-right shift" kernel? Do you expect a typically kernel classifier to correctly or incorrectly classify I_3 and I_4?

(e) Prove that k_{shift} is a positive semi-definite kernel function. **Hint:** Use the fact that k_G is positive semi-definite.

Problem 2: Neural Networks for Curve Fitting (15pts)

Consider the following 2-layer, feed forward neural network for single variate regression:

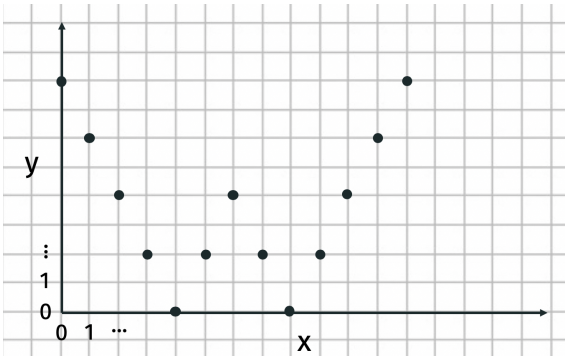


Let $W_{H,1}, W_{H,2}, W_{H,3}, W_{H,4}$ and $b_{H,1}, b_{H,2}, b_{H,3}, b_{H,4}$ be weights and biases for the hidden layer. Let $W_{O,1}, W_{O,2}$, $W_{O,3}, W_{O,4}$ and b_O be weights and bias for the output layer. The hidden layer uses rectified linear unit (ReLU) non-linearities and the output layer uses no non-linearity.

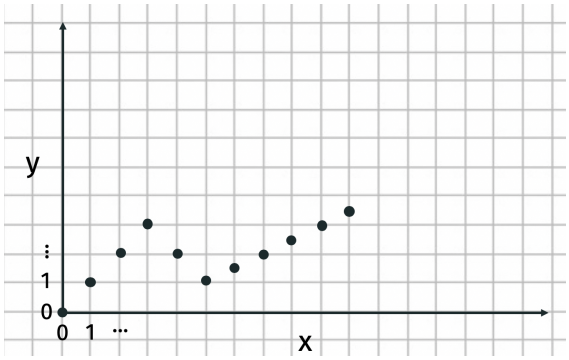
Specifically, for $i = 1, \ldots, 4$, $z_i = \max(0, \bar{z}_i)$ where $\bar{z}_i = W_{H,i}x + b_{H,i}$. And

$$y = b_O + \sum_{i=1}^{4} W_{O,i} z_i.$$

(a) For each of the two datasets below, determine values for weights and biases which would allow this network to perfectly fit the data.



Dataset 1



Dataset 2

(b) For input parameters $\boldsymbol{\theta}$ let $f(x, \boldsymbol{\theta})$ denote the output of the neural network for a given input x. We want to train the network under the squared loss. Specifically, given a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, we want to choose $\boldsymbol{\theta}$ to minimize the loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - f(x_i, \boldsymbol{\theta}))^2.$$

Write down an expression for the gradient $\nabla\mathcal{L}(\boldsymbol{\theta})$ in terms of $\nabla f(x_1, \boldsymbol{\theta}), \ldots, \nabla f(x, \boldsymbol{\theta})$. **Hint:** Use chain rule.

(c) Suppose we randomly initialize the network with ± 1 random numbers:

$$\begin{aligned} W_{H,1} = -1, W_{H,2} = 1, W_{H,3} = 1, W_{H,4} = -1 \\ b_{H,1} = 1, b_{H,2} = 1, b_{H,3} = -1, b_{H,4} = 1 \\ W_{O,1} = -1, W_{O,2} = -1, W_{O,3} = -1, W_{O,4} = 1 \\ b_O = 1 \end{aligned}$$

Call this initial set of parameter $\boldsymbol{\theta}_0$. Use forward-propagation to compute $f(x, \boldsymbol{\theta}_0)$ for $x = 2$.

(d) Use back-propagation to compute $\nabla f(x, \boldsymbol{\theta}_0)$ for $x = 2$. To do the computation you will need to use the derivative of the ReLU function, $\max(0, z)$. You can simply use:

$$\frac{\partial}{\partial z} \max(0, z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$

This derivative is discontinuous, but it turns out that is fine for use in gradient descent.

Problem 3: Triangulating Points via Principal Component Analysis (20pts)

(15 pts) Consider a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ arranged as rows in an $n \times d$ matrix $\mathbf{X}$. Assume $d \leq n$. As discussed in class, PCA can be used to find a set of shorter "code vectors" $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$ that are useful for data visualization. Specifically, if we choose these code vectors to be the loading vectors of PCA applied to $\mathbf{X}$ we should have that for each i, j:

$$\langle \mathbf{z}_i, \mathbf{z}_j \rangle \approx \langle \mathbf{x}_i, \mathbf{x}_j \rangle \qquad \text{and} \qquad \|\mathbf{z}_i - \mathbf{z}_j\|_2 \approx \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

What if we don't have access to $\mathbf{X}$ itself, but only to the *distances* between each pair of data points? I.e. you are given symmetric $n \times n$ matrix $\mathbf{D}$ with $\mathbf{D}_{i,j} = \mathbf{D}_{j,i} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$.

(a) From the information in $\mathbf{D}$ show how to recover $\mathbf{X}\mathbf{X}^T$ for a data set whose distances match those in $\mathbf{D}$. **Hint:** Since distances are only unique up to rotation and translation, you can assume that one of the points is centered at the origin. E.g. that $\mathbf{x}_1 = \mathbf{0}$.

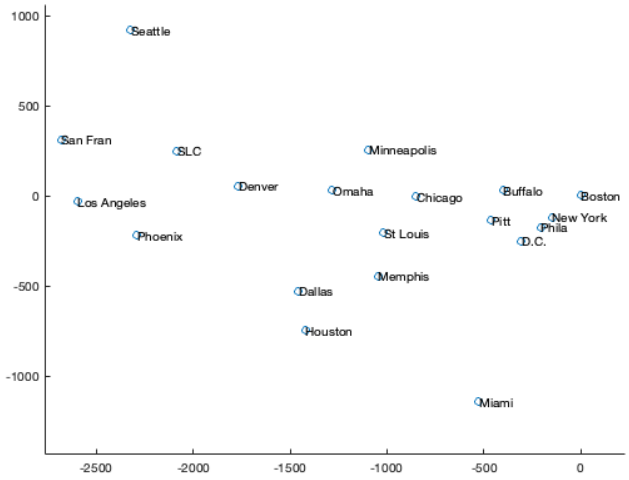
(b) Explain how to use the SVD of $\mathbf{X}\mathbf{X}^T$ to recover the rank k loading vectors of $\mathbf{X}$, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$.

(c) When we set $k = d$, prove that for all i, j we will obtain vectors that *exactly* capture the geometry of $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$. I.e. that:

$$\|\mathbf{z}_i - \mathbf{z}_j\|_2 = \|\mathbf{x}_i - \mathbf{x}_j\|_2$$

This is a really useful property – it means that if you have pairwise distances between points that you know come from a low-dimensional space, you can back out that arrangement using PCA. This problem is related to that of using triangulation to locate points based on distances.

(d) Implement your method from part (a) and (b) and run it on the U.S. cities dataset provided in `UScities.txt` for rank $k = d = 2$. This data set contains unsquared Euclidean distances between 20 cities, so you need to square the distances to obtain $\mathbf{D}$. Plot your estimated city locations $\mathbf{z}_1, \ldots, \mathbf{z}_n$ on a 2D plot and label the cities to make it clear how the plot is oriented. Submit these images and your code with the problem set (attached to the same PDF). I'm expecting a result similar to the one below (note that the locations look correct up to rotation).



(e) A cool property of this technique is that it's very robust to inaccuracies in the distance matrix $\mathbf{D}$. Rerun your code, but where you first perturb the distances in $\mathbf{D}$. Specifically, replace both $\mathbf{D}_{i,j}$ and $\mathbf{D}_{j,i}$ with a random number between $(1-r)\mathbf{D}_{i,j}$ and $(1+r)\mathbf{D}_{i,j}$ (i.e., keep the matrix symmetric). Plot the city locations recovered for a few different values of r. How high can you set the value and still get a good approximation to the city locations? This last answer doesn't need to be a quantitative.